Fragment Programming

Session 208

Fragment Programming

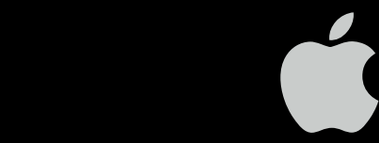
James A. McCombe
3D Graphics Engineer
tube@apple.com

Why Fragment Programming?

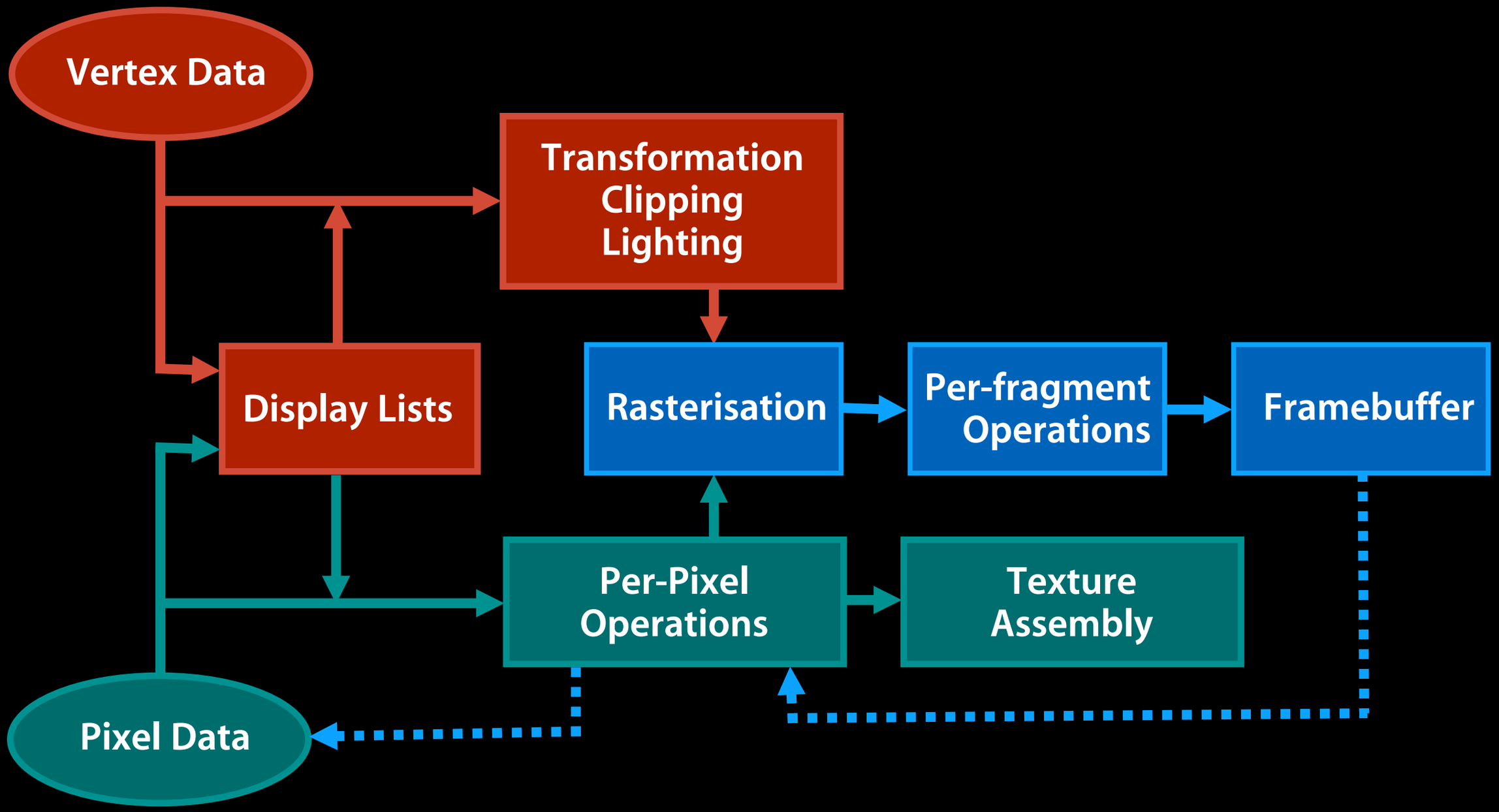
- Many applications
- Nominal performance penalty
- Apple provides best tools in the industry

What You'll Learn

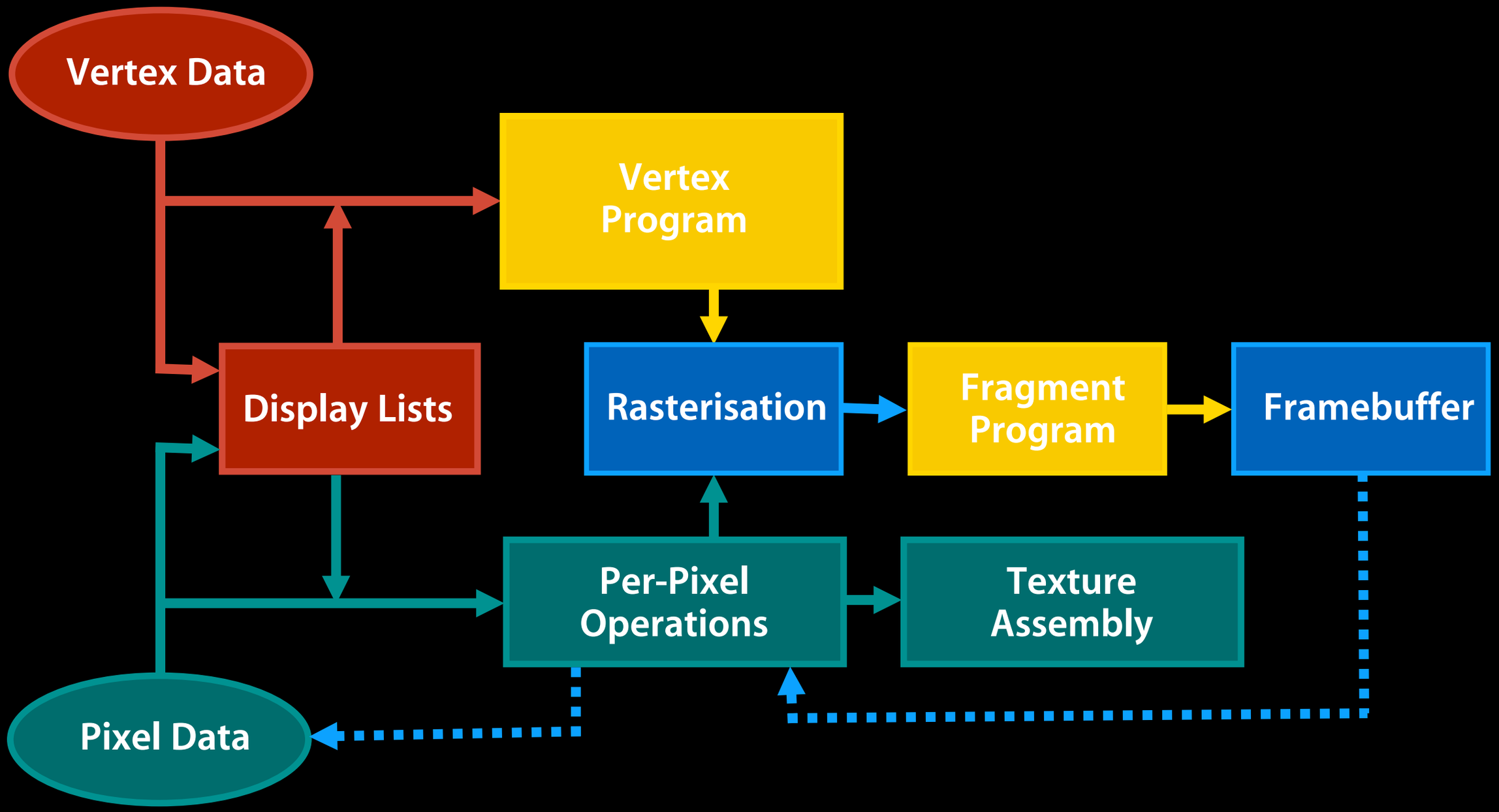
- Where it fits in the OpenGL rendering pipeline
- Supported graphics hardware
- The details on ARB_fragment_program
- Legacy hardware
- ShaderBuilder gets a face lift!
- The basics, working towards 2D image processing
- General purpose computation using multi-pass
- Optimisation and fragment programming
- Implementing a per-pixel lighting model



Where Does It Fit?



Where Does It Fit?



Supported Hardware

- ARB_fragment_program
  - ATI Radeon 9700
- ATI_text_fragment_shader
  - ATI Radeon 9700
  - ATI Radeon 9000
  - ATI Radeon 9000 Mobility

What Exactly Is a Fragment Program

- Ran once for every color lookup during rasterisation
- Outputs
  - Single RGBA color, optional depth
- Inputs
  - Interpolated texture coords, vertex color, window position
  - OpenGL state, program parameters
  - Texels sampled from texture units

ARB_fragment_program

Arithmetic

ABS	LIT	SIN
ADD	LRP	SUB
COS	MAD	XPD
DP3	MAX	
DP4	MIN	
DPH	MOV	
DST	MUL	
EX2	POW	
FLR	RCP	
FRC	RSQ	
LG2	SCS	

Logic

CMP
SGE
SLT

Texture Sampling

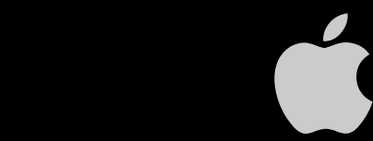
TEX
TXB
TXP

Miscellaneous

KIL
SWZ

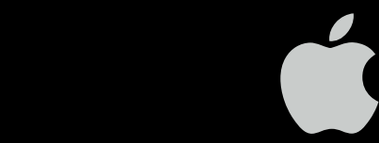
ARB_fragment_program

- Input fragment attributes
  - Color, Texture coordinate, Fog coordinate, Window position
- State parameters
  - Materials, Lights, Matrices, etc.
- Program parameters
  - Program specific, program independant
- Result attributes
  - Color, depth



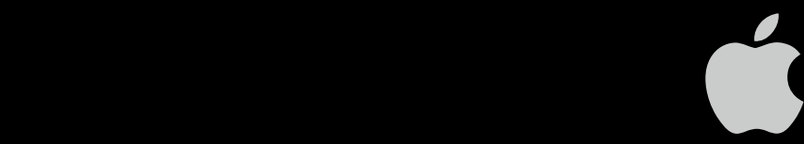
ARB_fragment_program Advantages

- Portability
  - Available on high-end ATI and is now approved by the ARB as the standard
- Rich instruction set
  - Allows more mathematically correct models
  - Less frustrating lookup table generation
- Flexible texture sampling
  - Multiple samples from same texture unit with different coordinates
  - Get more work done per rendering pass

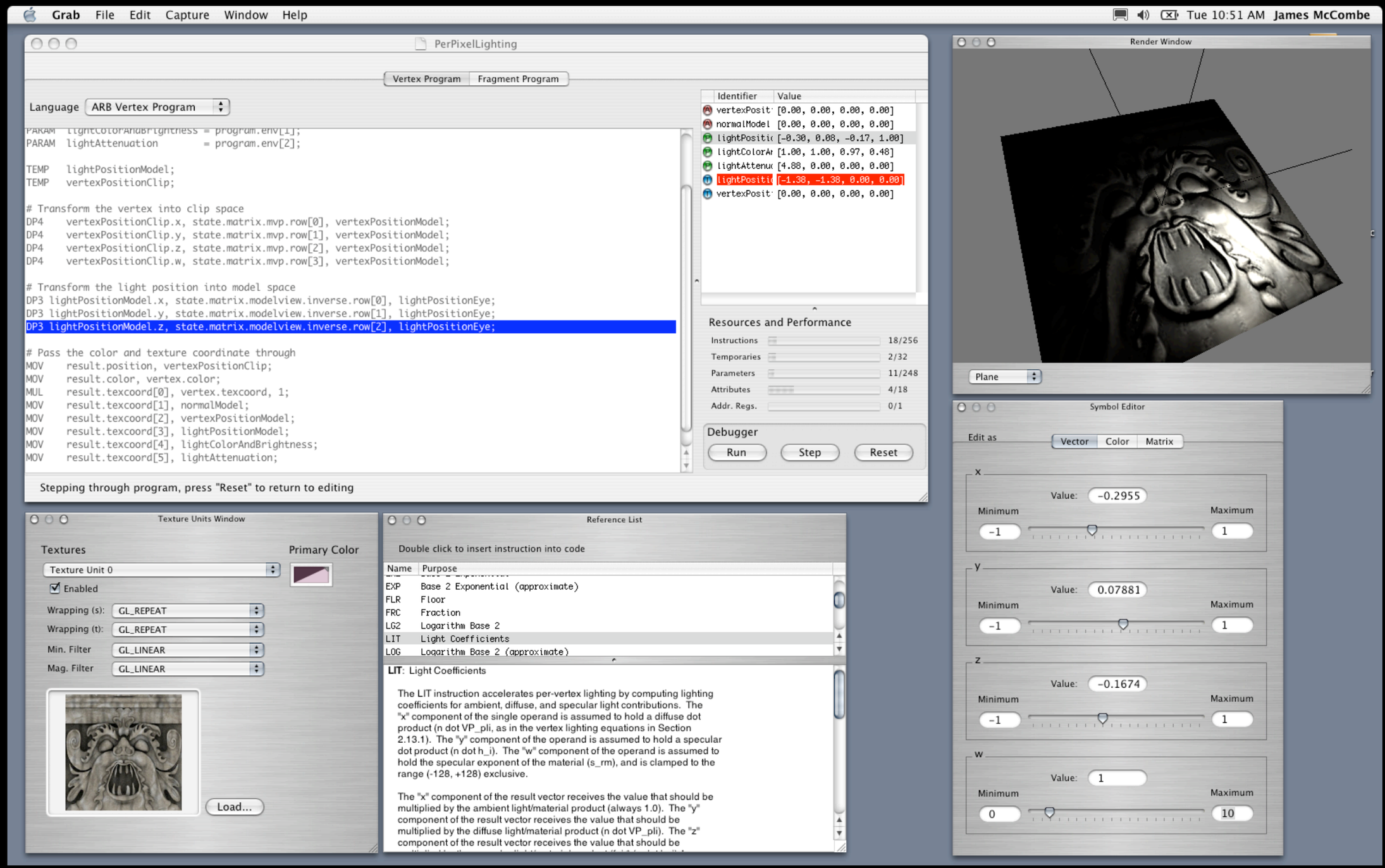


ATI_text_fragment_shader

- Vendor specific extension
- Allows basic fragment programming
- Lots of current hardware support
- ShaderBuilder supports this language
- Avoid this and focus on ARB_fragment_program if possible



New ShaderBuilder



New ShaderBuilder

- Added support for ARB_fragment_program and ATI_text_fragment_shader
- Added support for changing more texture parameters
- Program parameters can be changed on the fly
- Realtime Resource monitor
- Better solution for multi-head users
- Rework of the underlying code for better performance and more code sharing with the OpenGL Profiler
- Better documentation
- It’s available on the seed CD you get at WWDC

Demo

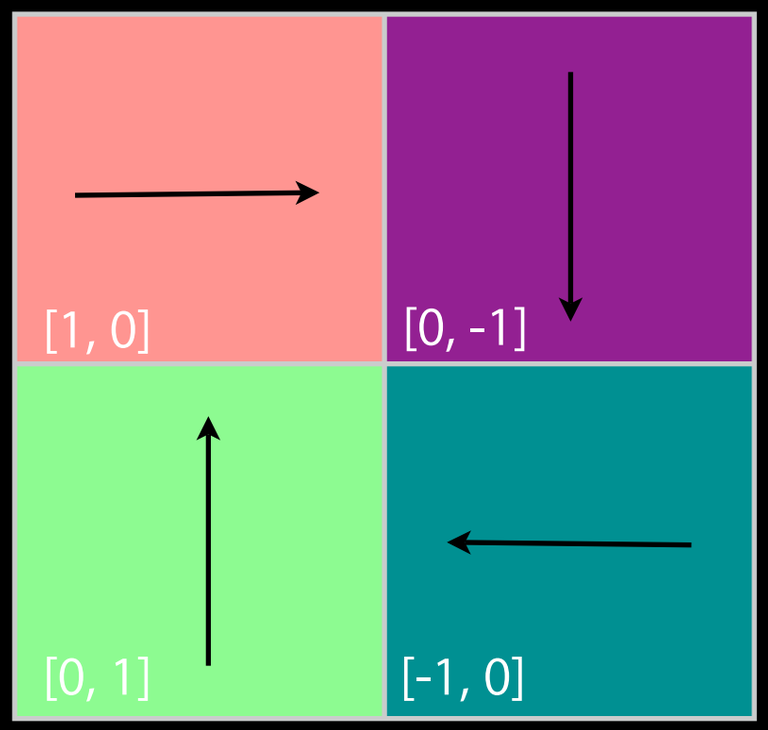
The Basics of Fragment Programming

2D Image Displacement

- Another example of using fragment program to offload work from your CPU
- Excellent performance
- Could make interesting inter-scene transitions in games

Displacement Map

- Flow vector encoded as color




```
color = (vector / 2) + 0.5
```

Demo

2D Image Processing

Game of Life!

- Cellular automata rule set conceived by Dr. John Conway in 1970

- It can be done using a fragment program
- Good example of how to use multiple pass rendering
- It runs obscenely fast

The Rules

- A cell is either alive or dead
- ‘c’ is the current cell
- n0-n7 are neighbour cells

n0	n1	n2
n7	C	n3
n6	n5	n4

N (number of neighbours) =
n0+n1+n2+n3+n4+n5+n6+n7

```
if (C == 0) {
   if (N == 3) C = 1;
} else {
   if ((N == 2) || (N == 3))
      C = 1;
   else
      C = 0;
}
```

“Life” in ARB_fragment_program

```
PARAM texelSize = { 0.00390625, 0.00390625 };
TEMP c0, c1, t0, t1, t2, t3, t4, t5, t6, t7;
TEMP r0, r1, r2;


MOV t0, fragment.texcoord[0];
ADD t0, fragment.texcoord[0], texelSize;
MOV t1, t0;
ADD t1.x, t0, -texelSize;
MOV t2, t1;
ADD t2.x, t1, -texelSize;
MOV t3, t2;
ADD t3.y, t2, -texelSize;
MOV t4, t3;
ADD t4.y, t3, -texelSize;
MOV t5, t4;
ADD t5.x, t4, texelSize;
MOV t6, t5;
ADD t6.x, t5, texelSize;
MOV t7, t6;
ADD t7.y, t6, texelSize;
```

“Life” in ARB_fragment_program

```
TEX c0, fragment.texcoord[0], texture[0], 2D;
TEX t0, t0, texture[0], 2D;
TEX t1, t1, texture[0], 2D;
TEX t2, t2, texture[0], 2D;
TEX t3, t3, texture[0], 2D;
TEX t4, t4, texture[0], 2D;
TEX t5, t5, texture[0], 2D;
TEX t6, t6, texture[0], 2D;
TEX t7, t7, texture[0], 2D;
```

```
MOV r0, t0;
ADD r0, r0, t1;
ADD r0, r0, t2;
ADD r0, r0, t3;
ADD r0, r0, t4;
ADD r0, r0, t5;
ADD r0, r0, t6;
ADD r0, r0, t7;
```

“Life” in ARB_fragment_program

```
SGE t2, r0, 3;
SLT t3, r0, 4;
MUL t2, t2, t3;

MUL r1, r1, c0;
MUL r1, r1, t0;
ADD r1, r1, t2;



MOV result.color.xyz, r1;
MOV result.color.w, 1;
```

Establishing Feedback

- Setup the read buffer with glReadBuffer()
- Use glCopyTexSubImage2D()
  - More likely to be fully accelerated than glCopyTexImage2D()
  - The data won't even leave the graphics card—direct VRAM -> VRAM copy

```
glReadBuffer(GL_FRONT_LEFT); /* Could use aux buffer */
glCopyTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, 0, 0,
                    512, 512); /* Buffer width, height */
```

Demo

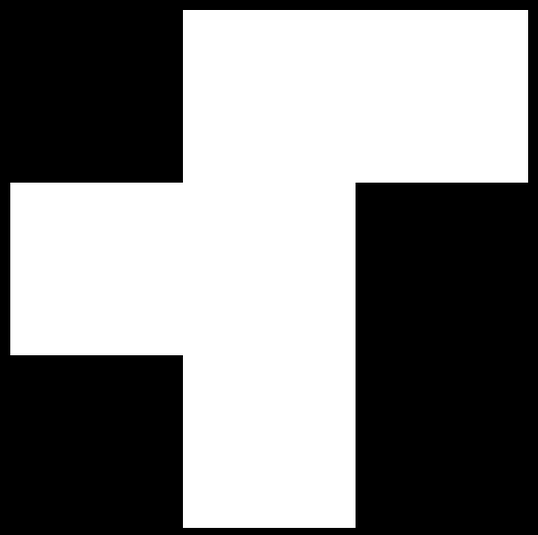
“Life” Running On an ATI Radeon 9700

Optimisation

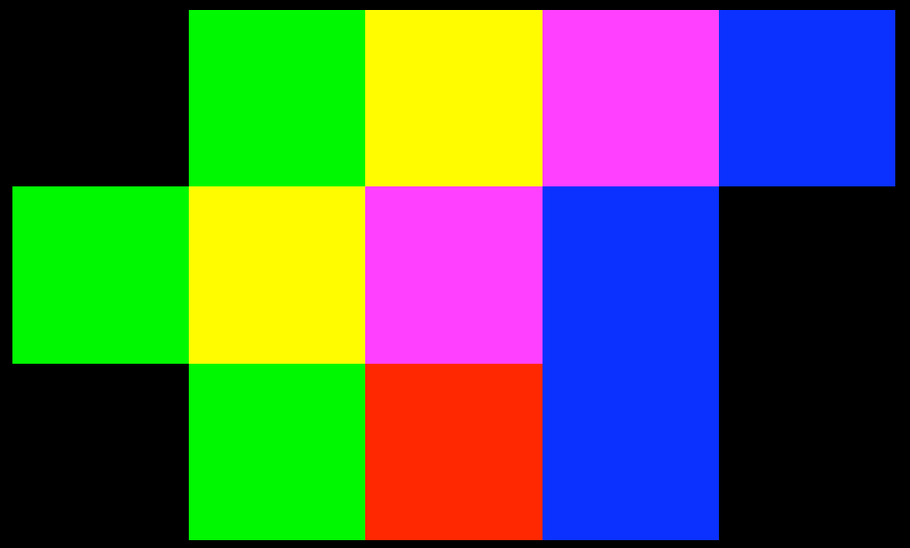
- Offload as much calculation from per-fragment to per-vertex as possible
- Lookup tables as textures are your friend
- Make sure you're not missing a nice instruction that does the same work that you're currently doing with 2 or more instructions!

Optimising “Life”

- Less instructions and texture samples to calculate total number of living neighbours



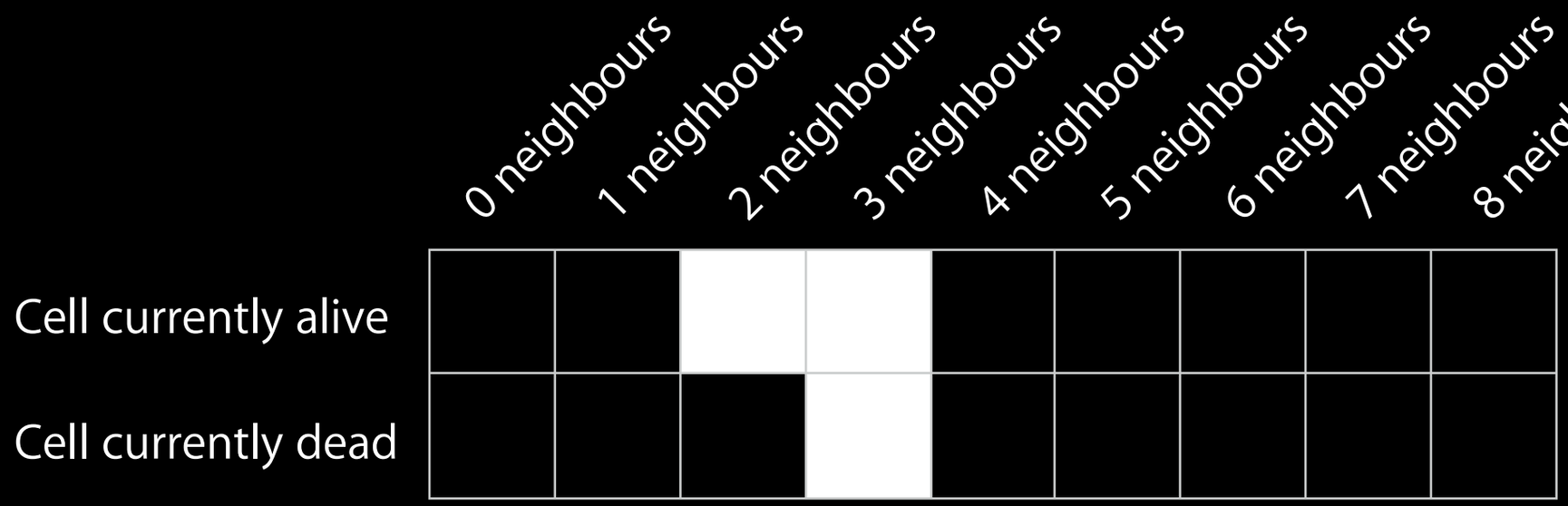
Typical R-pentomino life configuration



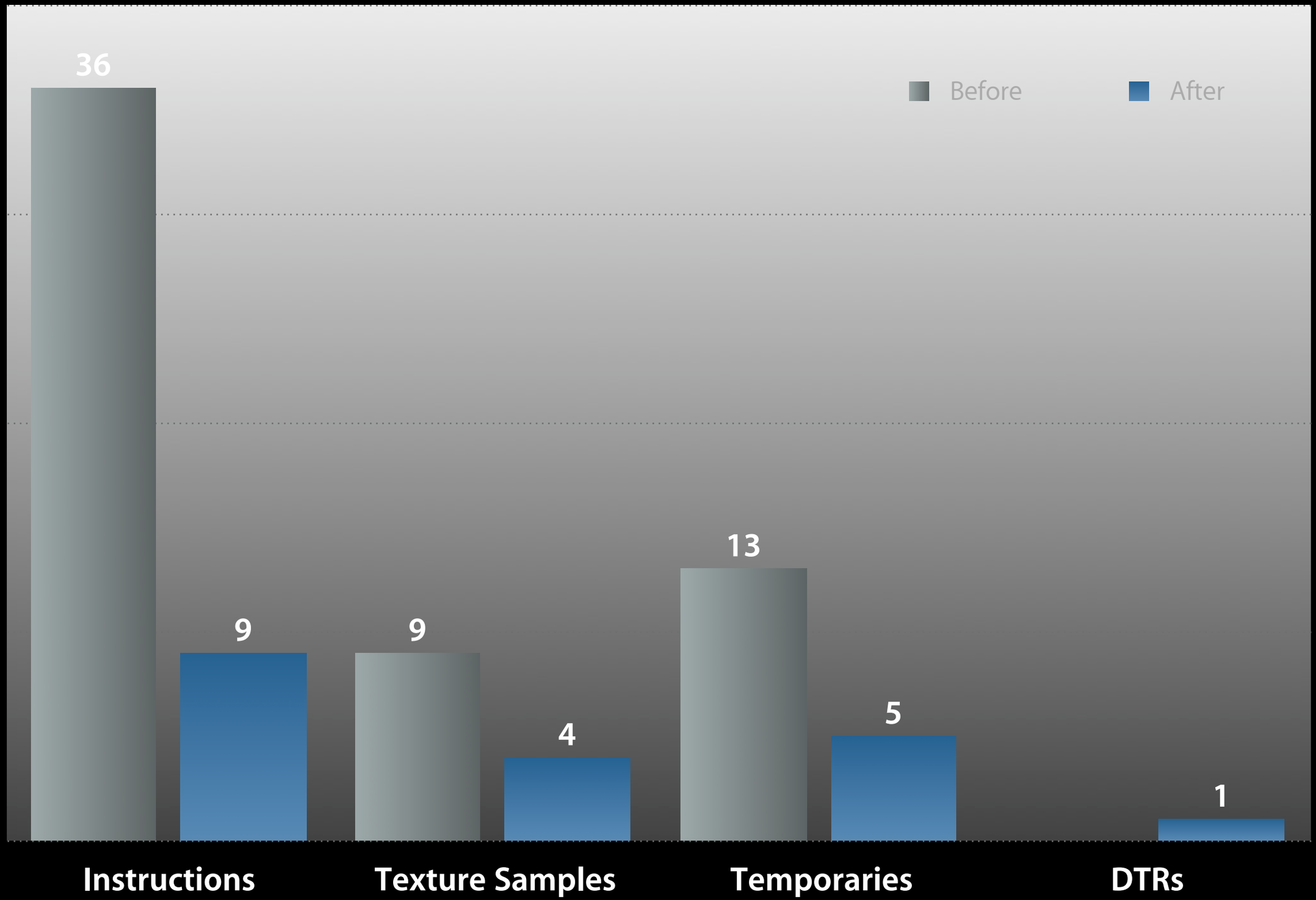
Pack neighbour cell states into the r, g, b channels allowing more calculation to be done per instruction

Optimising “Life”

- Use a lookup texture to implement the life logic instead of a series of instructions



Results of Optimisation



Per-pixel Spot Lighting Model

- Contributing entities are:
  - Light position (Lp)
  - Light brightness (Lb)
  - Fragment position (Fp)
  - Fragment normal (Fn)
  - Polygon normal (Pn)
- All calculated in model space

Normal Map

Gargoyle face base texture

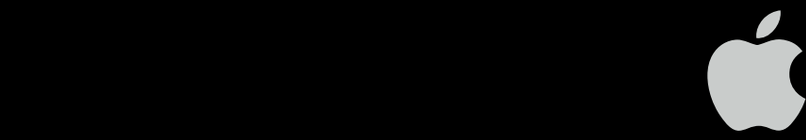
Per-pixel normals for a gargoyle face

- Normal vector stored as color
- Normal must be “unpacked” since colors are stored only in positive space

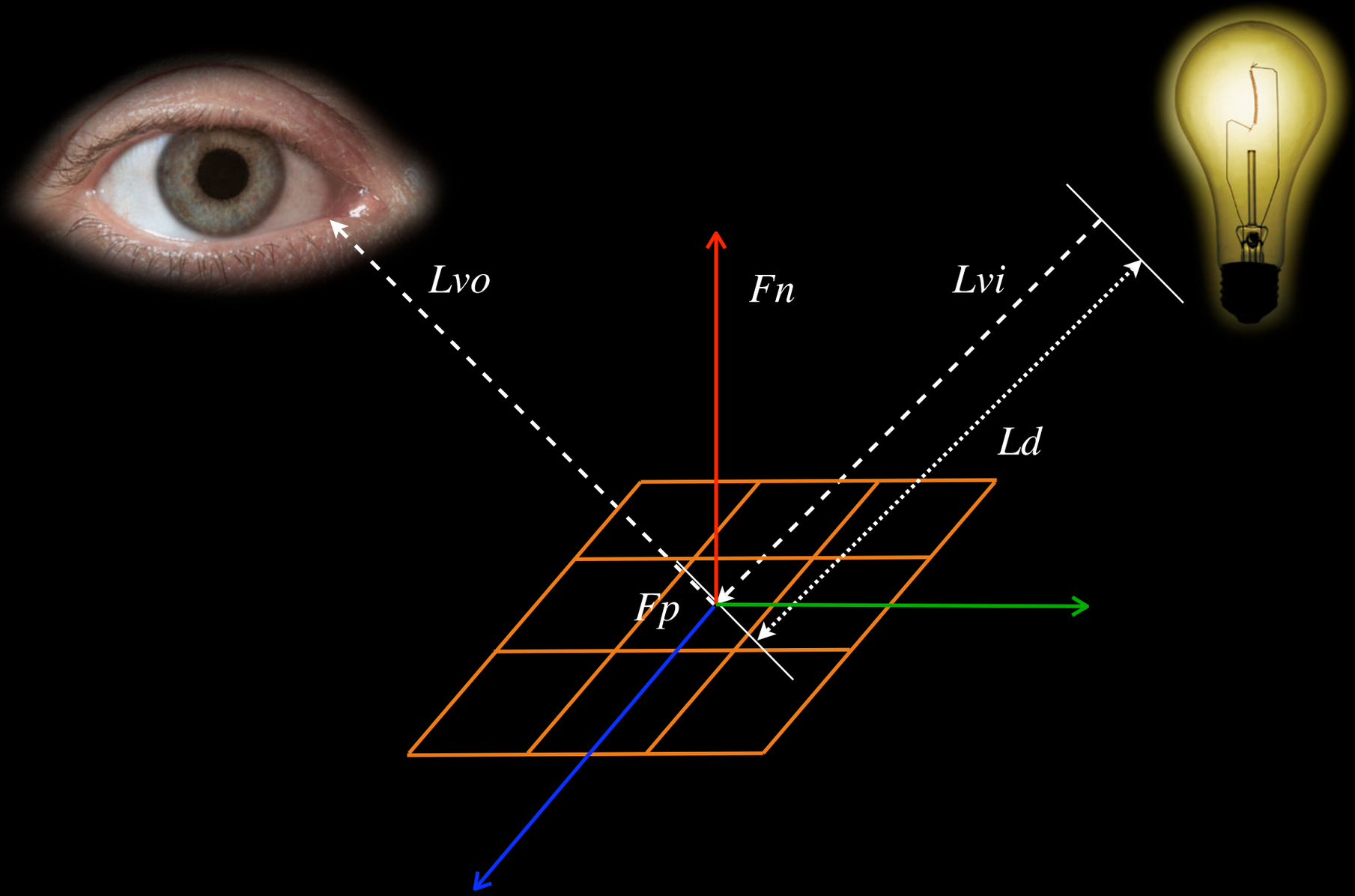
```
normal = ((color * 2) - 1)
```

Per-pixel Spot Lighting Model (Cont.)

- Calculate the vector from light to fragment (Lvi)
- Calculate the distance from light to fragment (Ld)
- Acquire the fragment normal (Fn)
- Transform Fn relative to polygon normal (Pn)
- Calculate light vector after reflection (Lvo)
- Dot product incident light ray with light direction
- Attenuate light intensity scalar with function of Ld
- Modulate with light brightness



Per-pixel Spot Lighting Model III



Demo

Per-pixel Lighting With ShaderBuilder

Who to Contact

James A. McCombe
3D Graphics Engineer
tube@apple.com

Geoff Stahl
3D Software Engineer
gstahl@apple.com

John Stauffer
OpenGL Engineering Manager
stauffer@apple.com

http://developer.apple.com/wwdc2003/urls.html

Q&A

James A. McCombe
3D Graphics Engineer
tube@apple.com

http://developer.apple.com/wwdc2003/urls.html

WWDC 2003

Worldwide Developers Conference